Association for Information Systems
AIS Electronic Library (AISeL)

PACIS 2013 Proceedings

Pacific Asia Conference on Information Systems (PACIS)

6-18-2013

Towards Next Generation Business Process Model Repositories – A Technical Perspective on Loading and Processing of Process Models

Hanns-Alexander Dietrich
University of Muenster – ERCIS, hanns-alexander.dietrich@ercis.uni-muenster.de

TOWARDS NEXT GENERATION BUSINESS PROCESS MODEL REPOSITORIES – A TECHNICAL PERSPECTIVE ON LOADING AND PROCESSING OF PROCESS MODELS

Hanns-Alexander Dietrich, University of Muenster – ERCIS, Muenster, Germany,
hanns-alexander.dietrich@ercis.uni-muenster.de

Abstract

Business process management repositories manage large collections of process models ranging in the thousands. Additionally, they provide management functions like e.g. mining, querying, merging and variants management for process models. However, most current business process management repositories are built on top of relation database management systems (RDBMS) although this leads to performance issues. These issues result from the relational algebra, the mismatch between relational tables and object oriented programming (impedance mismatch) as well as new technological developments in the last 30 years as e.g. more and cheap disk and memory space, clusters and clouds. The goal of this paper is to present current paradigms to overcome the performance problems inherent in RDBMS. Therefore, we have to fuse research about data modeling along database technologies as well as algorithm design and parallelization for the technology paradigms occurring nowadays. Based on these research streams we have shown how the performance of business process management repositories could be improved in terms of loading performance of processes (from e.g. a disk) and the computation of management techniques resulting in even faster application of such a technique. Exemplarily, applications of the compiled paradigms are presented to show their applicability.

Keywords: Business Process Model Repository, Algorithm Design, Database Design, NoSQL.

1 MOTIVATION

More and more organizations begin to document their core business procedures using business process models, in order to document, redesign, simulate, execute or control them (Vom Brocke et al. 2011). All these activities can be subsumed under Business Process Management (BPM). These models are stored in so called business process model repositories (BPMR), which can contain thousands of business process models (Becker et al. 2012a; Dijkman et al. 2012; Yan et al. 2012).

From a historical point of view, repositories were invented as a reusable piece of software, which incorporates standard operations and function for programmers and/or users that go beyond the functionalities of databases (Bernstein & Dayal 1994). Functionalities provided by databases are especially integrity, concurrency and access control. Whereas a repository adds functions as "checkout/checkin, version and configuration control, notification, context management, and workflow." (Bernstein & Dayal 1994) These general repositories can furthermore be extended or specialized to a domain like BPM, where these are enriched to provide special functionalities.

According to Dijkman et al. (2012) and Yan et al. (2012) the tasks of business process model repositories are widespread (cf. Figure 1). Building upon a central database, that stores the business models, business process repositories implement various management techniques like e.g. mining, querying, re-use, similarity search, refactoring, merging and variants management. A detailed description of these techniques is given by Dijkman et al. (2012). However, surveys revealed that the majority of BPMRs miss certain core repository functionalities or management techniques, required by practice (Shahzad et al. 2009; Yan et al. 2012).

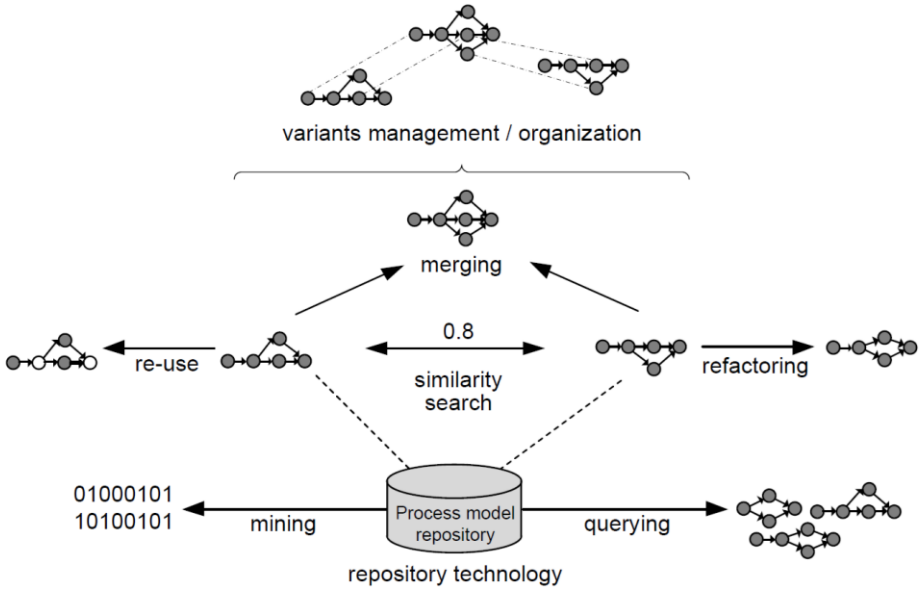

Figure 1. Repository technology and various management techniques for business process models (Dijkman et al. 2012)

As mentioned above the business process model repositories can contain thousands of business process models and should therefore incorporate scalability. When one of these process model management techniques is employed on such a big repository, the performance is of crucial importance. Otherwise the management techniques are not applicable in a reasonable time, which could hinder the practical relevance. Therefore, the various process model management techniques have to consider performance. Model querying, for instance, is used for runtime or design time compliance checking (Becker et al. 2012b). When runtime compliance checking is employed the amount of processes can be very high as every instance of a process has to be checked. Whereas when

design time compliance checking is employed the complexity of the processes rises as the integration of the fragmented process models into a whole enterprise process model can be necessary.

The de facto basis for repositories is a relational database management system (RDBMS), even for BPMR (Yan et al. 2012). RDBMS have been designed more than 30 years ago and have especially been built to make the most of the hardware and software paradigms that were available back then (Stonebraker et al. 2007). In recent years, these paradigms have been shifted to new ones, as e.g. disk space as well as memory is getting larger and cheaper. Furthermore, central processing units (CPUs) are faster and consist of more than one independent core (multi-core processor). Additionally, the cost of commodity computers has sunk so much that it is cheaper to buy several commodity computers and adapt the software to run in a clustered environment than to build one high performance system (Leavitt 2010). In a last step, nowadays it is possible to rent as many virtualized computers in a cloud as we can pay.

However, the relational databases have not really adapted to these new paradigms (Leavitt 2010; Stonebraker et al. 2007). RDBMS suffer from scaling problems, the complexity to convert the data into tables, and a huge feature set, which is not necessary in all scenarios (Leavitt 2010). In addition, the fixed relational schema as well as the usage of recursive queries can be problematic. Consider, for instance, a loading procedure in which a SQL query loads a process model in order to avoid multiple roundtrips to the RDBMS. The relational database schema for the model itself, the vertices and the edges between the vertices is shown in Figure 2. Loading the model with all its vertices is no problem, but including the edges in one SQL query too results into a problem of recursion. This is due to the fact that SQL has to recursively traverse the model to reconstruct its structure. Depending on the number of edges contained in a given model, this requires joining the vertex and edge tables for a potentially huge number of times which significantly slows down performance. The relational algebra is not capable of recursion and therefore the SQL 3 standard was extended to incorporate recursion in views (Libkin 2003). However, recursive views are still an extremely expensive operation (Libkin 2003).

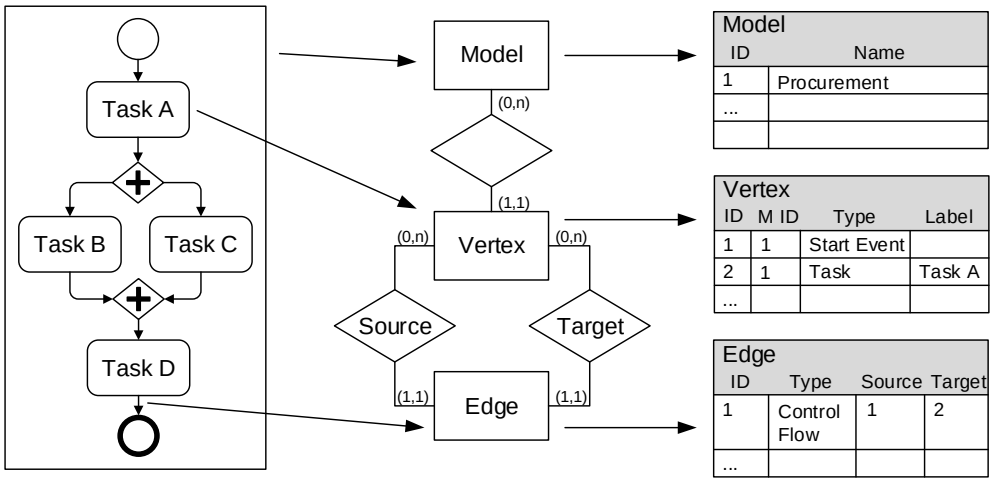

Figure 2. Mapping of a process model to a relational database schema

Against the backdrop of problems relational database systems exhibit with graph or graph-like data structures as well as the technological changes of recent years, we want to show how new approaches for storing process models that are based around NoSQL can be applied to solve two main problems. On the one hand, we want to increase the loading performance of processes from business process model repositories. On the other hand, we want to accelerate the execution of the management techniques by showing some starting points for parallelization. For a management technique the overall execution time ($T_{Management\ Technique}$) consists of the time spent for loading the model from the model repository into memory (T_{Load}) and the actual application of the algorithm to the model ($T_{Computation}$) resulting in: $T_{Management\ Technique} = T_{Load} + T_{Computation}$.

The remainder of this paper is structured as follows. In Section 2, we discuss the related work on BPMR. In Section 3, we proceed with current paradigms that can be employed to increase the loading performance as well as the computation performance. In Section 4, these paradigms are reflected for their usage in a next generation BPMR. Furthermore, we provide some consideration on the implementation of management functions in Section 5. In Section 6 we briefly describe some prerequisites and limitations of our approach for storing process models. Section 7 provides a conclusion and an outlook to future research.

2 RELATED WORK

The requirements of a business process model repository have been reviewed by Shahzad et al. (2009). They combine essential requirements of (process) repositories with requirements from a set of use cases and evaluate them against existing process repositories. From six process repositories only one e.g. is capable of version management. In a next research step they have elicited the requirements of a business process model repository from a stakeholder‘s (researchers and practitioners) perspective (Shahzad et al. 2010). The analysis revealed 8 requirements centered on model reuse.

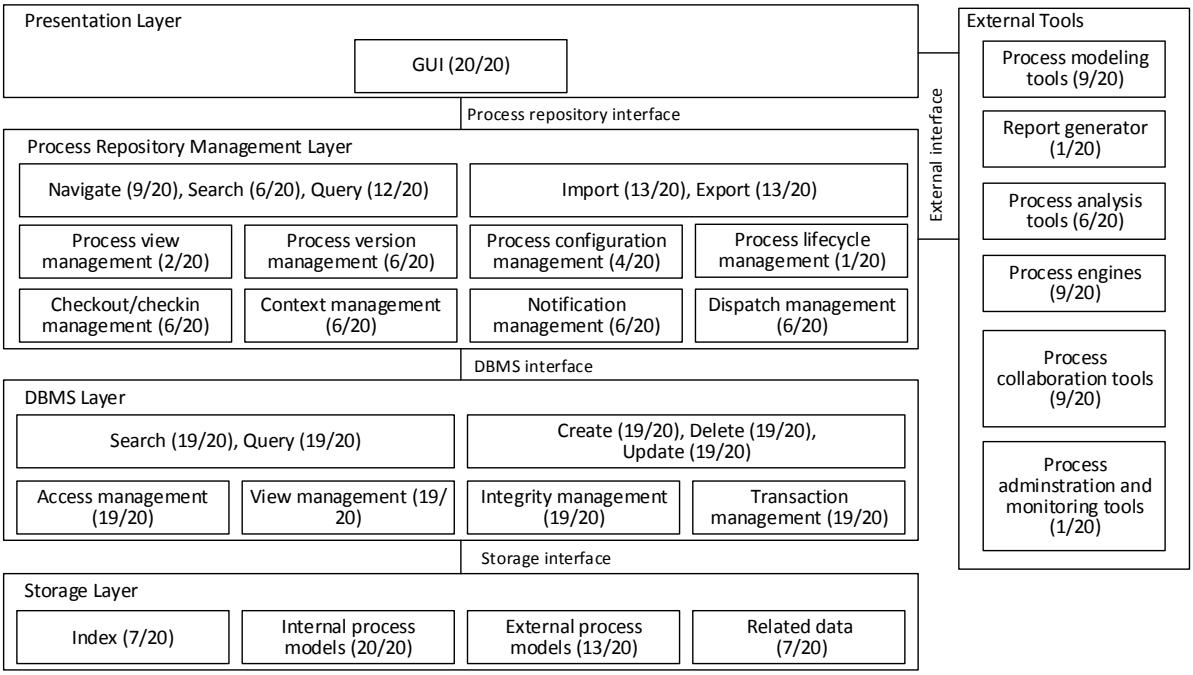

Figure 3. A descriptive business process model repository reference architecture and the overview of the functionalities implementation ratios according to Yan et al. (2012)

Yan et al. (2012) have constructed a classification framework for business process model repositories and applied it in a survey. In the analysis, 20 business process model repositories were included, from which three motivations for the construction of them were drawn. Firstly, the business process model repositories were constructed to store and then allow for navigating, searching and querying models. Secondly, the models as well as the running instances of the models were stored for the purpose of interaction with a workflow engine. Thirdly, the development of a business process repository was motivated by re-using the stored models in form of complete process (reference) models or only parts of them. Furthermore, they have identified a gap between the functionalities of existing business process model repositories and current research. The analysis revealed that the basic functionalities in the DBMS layer (cf. Figure 3) like e.g. access management and create, read, update and delete (CRUD) operations are supported by all repositories, while advanced functionalities like e.g. process view management and process lifecycle management are neglected. Nevertheless, six business process model repositories were capable of version management (see Figure 3). However, they state that their

reference architecture is just a descriptive rather than a prescriptive one and therefore should not be used to actually structure a BPMR in such a way. Based on their findings (cf. Figure 3) they conclude that "there is much room for implementing advanced functionality for managing large collections of business process models, by using […] [business process] Model Repositories." (Yan et al. 2012)

For reasons of brevity, we refer to (Yan et al. 2012) for the description of the 20 business process repositories and are only introducing new or not included ones. Eid-Sabbagh, Kunze, Meyer, & Weske (2012) have created an analysis platform for process models that could be reused by other researchers and allows them to concentrate on their actual research problem. The platform provides functionality to import a large variety of process models and is open for extensions. It does not provide any functionality to create or edit a process model. For analysis purposes it provides process metrics and can cluster the process models maintained in it. To allow for extensions the pipes and filters integration pattern (Hohpe & Woolf 2010) was used. It decomposes a larger task into independent and smaller processing steps and connects them with so called pipes (Hohpe & Woolf 2010). By doing so, the repository allows for reusing existing filters or writing new ones, and in doing so, extending it. Exemplarily, an integrated filter can extract all tasks of a BPMN process and another consecutive one extracts the labels of the given BPMN tasks.

3 CURRENT PARADIGMS

3.1 Aggregate Data Models

Relational DBMS organize their data into relations and tuples, more commonly known as tables and rows. A relation is a set of tuples and each tuple consists of a set of name-value pairs representing a row. Based on these relations they provide relational algebra that allows for operations that utilize relations as input and output. These mathematically elegant concepts also have drawbacks, when they are employed in, for instance, object-oriented programming languages leading to the so called impedance mismatch (Sadalage & Fowler 2012): The in-memory data structures of object-oriented programming languages differ from the relational model. If, for instance, the receipt data structure contains a list of ordered products, these nested records cannot directly be mapped on a single relational model (cf. Figure 4). In fact, the nested data structures have to be divided into own relations. This is due to the value of a tuple, which can only represent a simple −not nested− value.

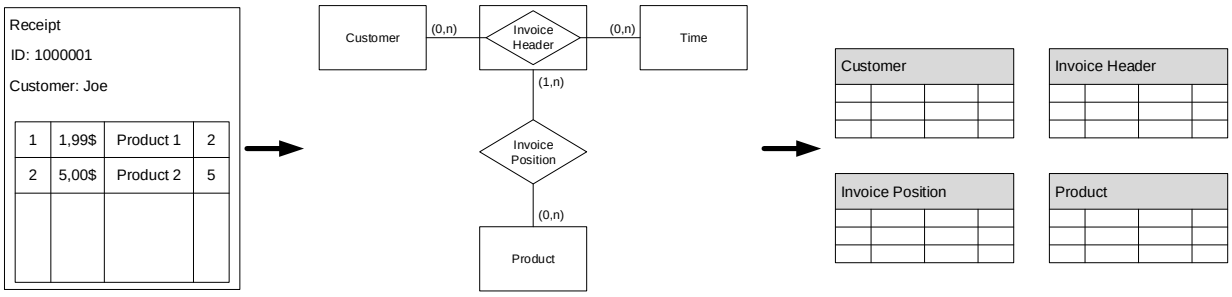

Figure 4. Transformation of the receipt structure from an object-oriented data structure to a relational model

Against the backdrop of the impedance mismatch, new database systems have emerged to overcome this problem in specific application domains. These are key-value databases, document databases, column-family stores and graph databases. The first three databases are aggregate oriented database systems. In contrast to tuples, aggregates allow for more complex data structures as for example nested records (Sadalage & Fowler 2012). The term "aggregate" was coined by Evans (2004) and represent a collection of related objects, which should be treated as one unit for data manipulations

and consistency. The receipt data structure in Figure 4, for instance, is an example of such an aggregate. In most application scenarios the receipt as well as the invoice positions are accessed at the same time. While in relational databases this would result in join operations between the corresponding tables, the aggregate oriented database systems return the result without such time-consuming operations in between. However, different application scenarios can require different aggregates, for example if we want to access all receipts of one customer. A solution can be to store this second aggregate in the database, or construct it in the database from all the receipt aggregates mentioned before. Another way would be to store a customer aggregate linking to the receipt aggregates. Altogether, the usage of aggregates requires a sophisticated and use-case reflected aggregate data model.

3.2 Aggregate Oriented and Graph Databases

Key-value databases store pairs of a key (or ID) and a value similar to a hash table (Sadalage & Fowler 2012). For the database the value is just a meaningless representation of bits like the binary large object (BLOB) data type in relational databases. To access the stored data a lookup with the primary key is always used, which generally has acceptable performance. As mentioned before, key-value databases are aggregate oriented meaning in the specification phase of an according database usage, the data model is defined by aggregates rather than simple data types. Besides the general size limit of the value, it is possible to store arbitrary data.

The document database stores pairs of a key and a document (Sadalage & Fowler 2012). However, compared to a key-value database the document database can use the structure of the aggregates (document) so the aggregates stay examinable. The documents are most likely serialized in a XML or JSON format and consist of collections, maps and primitive values. In contrast to a key-value database it is possible to build an index on the fields of a document, to retrieve parts of the aggregate or to simply query the documents based on a field.

The column-family stores differ from RDBMS in that they do not store the records in a row and therefore each value of the row after the other on disk, but they store groups of columns together (Sadalage & Fowler 2012). To describe their structure, one could say that they consist of two-level aggregates. These are similar to key-value stores, as the first level aggregates consist of a key and a second level aggregate (value). The key is used to identify the aggregate of interest, like a row identifier in a RDBMS. The second level aggregate is itself a map consisting of the column names and values. The second level aggregates are referred to as column families and grouped together in the aggregate to build a unit of access. The assumption behind it is that the values of a column family are accessed together.

Graph databases have been developed to overcome the problems of RDBMS with graph-like data structures (Sadalage & Fowler 2012). This is the situation, when the data has complex interconnections, like, for example, in social graphs. The structure or data model of a graph database is very simple, as it only consists of vertices connected by edges. The vertices and edges can have attributes to store values. In contrast to RDBMS, graph databases need no join operations to navigate through the graph. They store references between the vertices and edges that make a traversal very fast.

3.3 Algorithm Design

Algorithm design is about creating algorithms that solve a specific computational problem and furthermore work correct, efficient and are easy to implement (Skiena 2008). However, not all goals must be achievable at the same time (Skiena 2008). In order to develop an efficient algorithm for a specific problem, first we have to identify the core of the problem and then choose an appropriate algorithm design technique (Kleinberg & Tardos 2005). These algorithm design techniques provide generic patterns to solve various problems, like the more specific design patterns in software development. However, they are mostly applied to problems where a calculation on a large number of

combinatorial possibilities has to be solved (Kleinberg & Tardos 2005). The term efficiency in the area of algorithm design is not consistently defined, but mostly encompasses the required computing time as well as (memory) space. In addition, efficiency could also incorporate sustainability as e.g., energy efficiency, which is continuing to gain importance. Independently from the actual definition of efficiency and therefore the aim of the algorithm (minimize, for instance, computing time, memory space, energy consumption), the algorithm design techniques can be used as a starting point. A brute-force algorithm conducts search over the whole space of possible solutions and is therefore inefficient. The following algorithm design techniques therefore try to reduce the search space.

The most natural algorithm design technique are greedy algorithms (Kleinberg & Tardos 2005). The algorithm tries to optimize a criterion by grabbing the "best" thing in each step and thereby builds up a solution (Kleinberg & Tardos 2005; Skiena 2008). However, for the most problems no greedy algorithm exists, but when one exists it's possible to design many different greedy algorithms.

Another algorithm design technique is divide and conquer. A divide and conquer algorithms takes the input and decomposes it into smaller parts until it is small enough to be easily computable and then solves the problem recursively for each part (Kleinberg & Tardos 2005). The solutions of each part are combined again until a complete solution on the top level is reached (Kleinberg & Tardos 2005). A well-known example for a divide and conquer algorithm is merge sort (Skiena 2008).

3.4 MapReduce

MapReduce enables a kind of distributed divide and conquer programming model, which can be used to distribute calculations over processors or even computers (Dean & Ghemawat 2008). Historically, the development of MapReduce started with the rise of multiple computers in clusters, where the processing and storage of data had to change to account for scalability (Sadalage & Fowler 2012). With only one database, the only choice to make was to install the processing logic in the database or the client. Installing it on the client site gives the flexibility to choose the programming language, but comes at the cost that all processed data has to be loaded from the database into the client. Whereas installing the logic in the database the loading of the data is more efficient and only the processed data has to be transferred to the client, but the programming language is predefined by the database. Having multiple computers, the computation should be spread over all of them to gain as much performance as possible. On the other hand the amount of data to be transferred between them should be kept as small as possible. To handle this balance the MapReduce paradigm (Dean & Ghemawat 2008) can be employed, which is a form of the more general Scatter-Gather pattern (Hohpe & Woolf 2010). MapReduce allows spreading the calculation across multiple machines while holding the required data as well as the required logic on the same machine and therefore reduces the transfers between computers. For reasons of brevity, we show only the general concept and refer to Dean & Ghemawat (2008) for a detailed description.

A simple example in the area of business process model repositories is planning of human resources based on process models. Consider, for instance, a process which is performed 120 times a month and in the corresponding BPMN model the throughput time of each task as well as the executing department are annotated. The MapReduce patterns can be employed to calculate the aggregated duration of a single process execution for each department. Therefore the MapReduce pattern uses the aggregate structure of a process model as exemplarily shown in Figure 5. The Map function takes an aggregate structure as input and outputs key-value pairs. In the example the input is the process model and the outputs are key-value pairs of the BPMN tasks. Each key-value pair has the task name as the key and the duration as well as the execution department in a map as values. This specific mapping has been defined in the map function. In the lower part of Figure 5 the map and reduce function to calculate the duration of a single process execution for each department is shown. The map function creates a key-value pair for each department with the name as the key and a map of the durations as the value. The reduce functions operate on key-value pairs with the same key and aggregate them, for instance, into one key-value pair. In the example the reduce functions sum up the durations and

aggregate them into a key-value pair for each department. The map functions are independent from each other so they can be parallelized and assigned to different computers for each process. The MapReduce framework takes care of assigning the map function for each process to the computers and afterwards collects all the results. Before providing the results to the reduce function the results are grouped by the key and each group is handed over to the reduce function. Therefore, the reduce function operates on key-value pairs of the same key making it easier to develop a reduce function. The usage of the MapReduce pattern can be even more stressed out by the fact that for one department not only one but hundreds of processes can exist. Then the MapReduce framework can assign each process to a map function spreading the work over all computers.

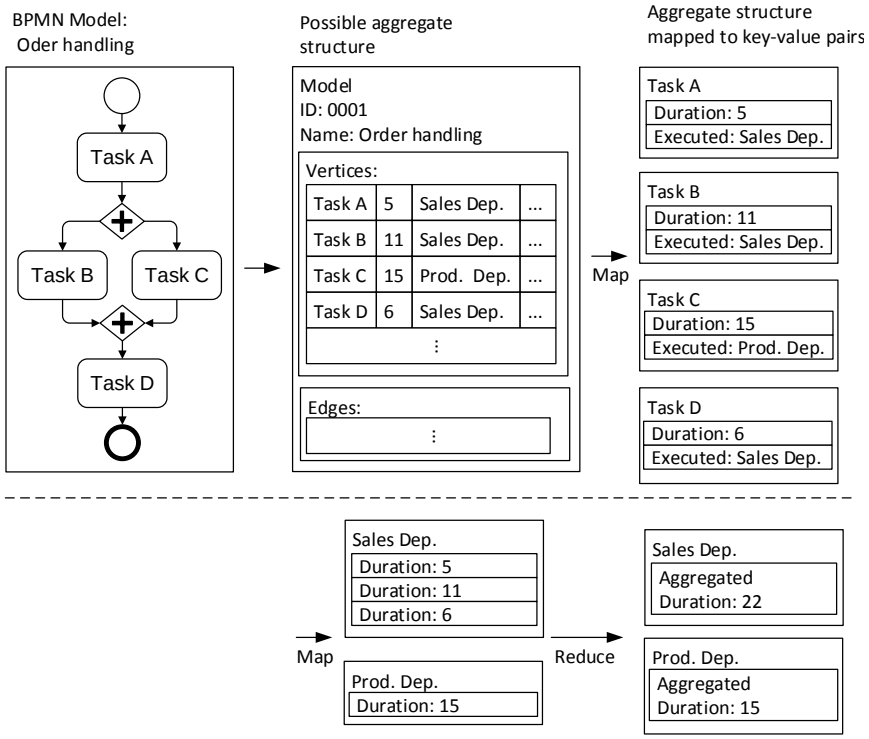

Figure 5. Example of a MapReduce pattern

4 NEXT GENERATION BUSINESS PROCESS MODEL REPOSITORIES

Against the backdrop of the current paradigms, we will highlight some drawbacks of current business process model repositories. Firstly, the scalability techniques of relational databases that are used by most of the current business process model repositories do not account for modern hardware developments. Secondly, we have an impedance mismatch between our used data structures and the employed relational storage mechanisms. Thirdly, the functions of relational algebra are not suited to process graph-like data structures. Fourthly, extended functions of business process model repositories, like, for instance, a process query, are not computed in a distributed way.

4.1 T_{LOAD}

In order to optimize the execution time of a management technique, defined as $T_{Management\ Technique} = T_{Load} + T_{Computation}$, in a business process model repository, we first concentrate on the loading time (T_{Load}) of a process: How can the loading time of a process be decreased against the backdrop of the aforementioned problems?

The general problem of the right choice of database technology was researched by Stonebraker et al. (2007). They come to the conclusion that one database cannot fit all problems and furthermore show that specialized database engines for various problems can outperform RDBMSs by 1-2 orders of magnitude. Additionally, Sadalage & Fowler (2012) argument for polyglot persistence, where not only a single database is used in the software architecture, but for each problem the best suited database is used and therefore added to the software architecture. With this in mind, we can construct next generation business model repositories with specialized databases for each problem (cf. Figure 6). Exemplarily, the session data of a user in such a repository can be handled by a key-value store. Furthermore, the process models can be serialized by the next generation business process model repository and stored in e.g. a document database. As the document database is aggregate capable, the impedance mismatch problem is solved. In contrast to a key-value database the document database allows for queries over the attributes of a model. In order to load a process from the document database, it is only necessary to load the complete aggregate stored under the key. As no views or multiple queries are necessary in contrast to RDBMSs, this will decrease the loading time (T_{Load}). Additionally, the time to store a new process model in an aggregate oriented database is also smaller, as only the complete aggregate has to be stored in contrast to several involved tables and constraints in RDBMSs. However, the choice of one or multiple databases is dependent on the actual use cases or supported functions of a BPMR.

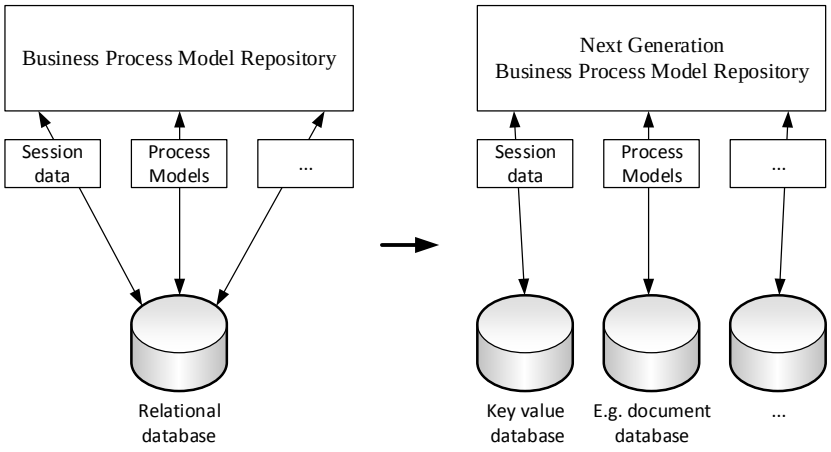

Figure 6. Architectural change of next generation business process model repositories for polyglot persistence

4.2 $T_{Computation}$

Having a starting point to decrease the loading time of processes, we have a closer look on the time required by the application of the management function's algorithm to the process model ($T_{Computation}$). Firstly, a decision has to be made between a fat (c.f. Figure 7 a) and thin client (c.f. Figure 7 b) and therefore where the business logic (management function's algorithm) is placed. With the usage of multiple computers in clusters or clouds, as it was discussed along with MapReduce, a fat client containing all the business logic without a distribution mechanism is not suitable. Even with a distribution mechanism, the stored data has to be transferred to each fat client generating a lot of traffic and a lag of time. Therefore, a thin client concept is appropriate. Furthermore, the distribution mechanisms, like for instance MapReduce, can be reused as they are integrated in some aggregate databases. This allows an easier development of parallel business logic inside the database, without the need to transfer the data. Additionally, some aggregate databases can be run in a cluster in order to improve the scalability and availability (Sadalage & Fowler 2012). Such a cluster setup allows distributing a MapReduce task to all computers forming the cluster (c.f. Figure 7 c). In summary, avoiding data transfers by incorporating the business logic into or near the database as well as

distributing the management function using MapReduce to several CPU cores or even computers will decrease the time of a management function's algorithm ($T_{Computation}$).

Apart from data transfers and distributed calculation, also the right algorithm design techniques should be considered. In order to implement the management functions of the repository, the management function itself or a sub problem can be efficiently solved with such a technique. However, first such an algorithm has to be designed for the problem at hand, which is already quite a complex task. Nevertheless, finding such an algorithm based on algorithm design techniques is no guarantee that it is also parallelizable as there is no direct relationship between these algorithms and parallel programming. There is a lot of research in the area of parallel programming taking place to create more suitable general algorithms design techniques for parallel programming. However, from the introduced algorithm design techniques only divide and conquer provides a starting point for parallelization. By employing divide and conquer a problem is decomposed into sub problems that can be solved on their own and therefore can be distributed by MapReduce. As many management functions for BPMR use partly graph algorithms (Becker et al. 2012a), the work of Lin & Schatz (2010) can be seen as a starting point providing design patterns for graph algorithms based on MapReduce.

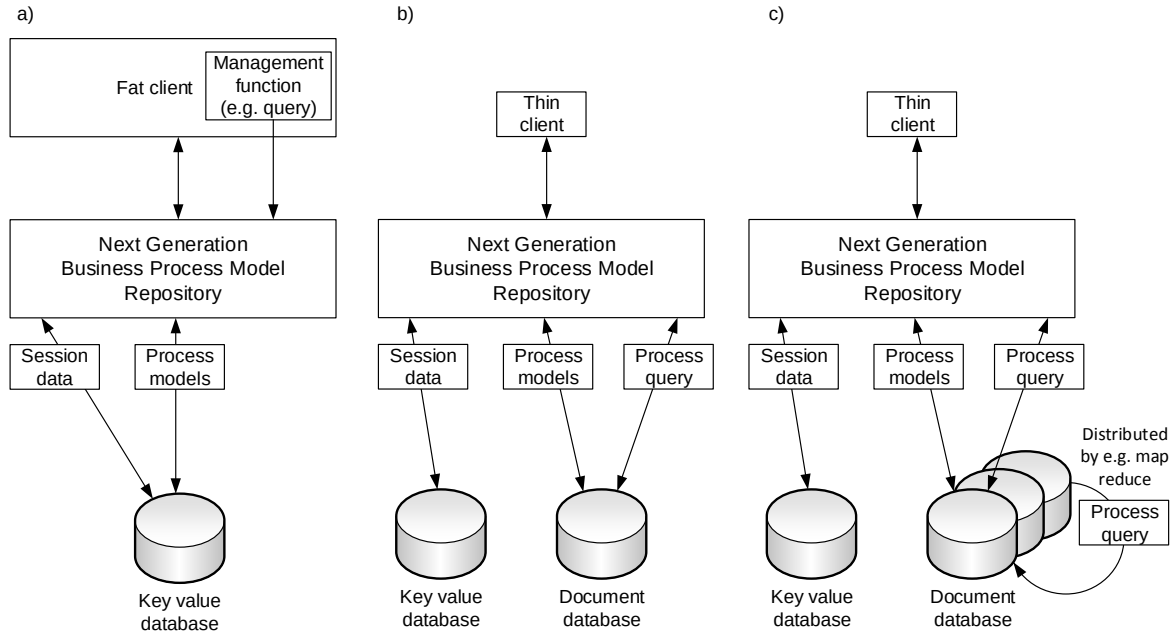

Figure 7. Exemplary architectures of business process model repositories with business logic implemented in different layers

Besides the general algorithm design techniques, also the caching and index techniques are important. Most managing techniques for business process model repositories in the literature incorporate caches or indexes to improve performance. Yan et al. (2005) show that for graph queries to detect patterns in process models utilizing indexes the query response time is: $T_{Query} = T_{Search} + |C_q| * (T_{Load} + T_{Computation})$. Firstly, they search for all index features in the query graph (T_{Search}) and construct a candidate set (C_q) from their indexes. Each candidate is loaded from, for instance, the disk (T_{Load}) and the pattern matching computation is performed ($T_{Computation}$). In doing so, they could speed up the graph query, as only relevant graphs have to be loaded from disk. In summary, index techniques can further improve the execution time of management techniques ($T_{Management\ Technique}$).

5 IMPLEMENTATION CONSIDERATIONS FOR MANAGEMENT FUNCTIONS

Different kinds of databases allow for specific features, which can be reused for the management techniques of the repositories. As we have seen in Section 2 the version management feature is underrepresented. Thinking in aggregates allows us , for instance, to store each version of a model including a timestamp as a separate aggregate and therefore implements a basic version management (cf. e.g. (Plattner 2009; Sadalage & Fowler 2012)). This is easily implementable with a key-value database, document database or column-family store.

Another example is model querying. In the literature several approaches for model querying exist like (Awad 2007; Beeri et al. 2006; Choi et al. 2007; Delfmann et al. 2010; Jin et al. 2010). One well-established approach is BPMN-Q (Awad 2007) that is specifically tailored for BPMN models. Furthermore, it has a visual query language to specify graph-based queries that in turn are transformed into SQL queries. The underlying relational database schema is described in (Awad & Sakr 2010):

BPModel(ModelID, ModelName, ModelDescription)

BPElements(ModelID, ElementID, ElementName, ElementType)

BPEdges(ModelID, EdgeID, SElementID, DElementID, EdgeType)

As explained before, recursive relational data schemas like BPEdges exhibit a bad loading performance, because of the necessary join operations (Sakr & Awad 2010). To overcome this problem they use sophisticated index techniques. Especially, to compute paths between nodes of a process model they use another table to store all possible paths (Awad 2007). This step has a high runtime of $O(N^3)$ and is therefore only conducted once (Awad 2007). In later versions they have improved it to only incrementally add paths based on their actual queries (Awad & Sakr 2010).

Graph databases are especially suited for such path queries. These could be implemented as a standard depth first search. Moreover, a depth first search can also be used to determine paths that shouldn't contain a certain process element. As there is no impedance mismatch and the graph database is highly optimized for such traversals, the table storing all paths could be omitted. The other sophisticated index techniques could be also transferred to a graph database. To our best knowledge this would result in lower execution times of BPMN-Q queries.

In Section 3.4 and 4.2 we have already shown how MapReduce could be used to spread calculations on process models over the connected computers. Exploiting the MapReduce pattern, BPMN-Q queries can be decomposed into independent Map and Reduce tasks. Furthermore, when one or more BPMN-Q queries are run against one or more process models, each pair of query and model can be run independently by MapReduce. However, to our best knowledge MapReduce is only integrated into several key-value databases, document databases and column-family stores.

6 PREREQUISITES AND LIMITATIONS

The main data model used here is a single process. However, if we consider current BPMRs they also maintain connections between models or even provide a highly integrated enterprise modeling framework. This integration leads to aggregates that share common elements making it a lot more complicated to maintain integrity between the aggregates if one is changed. However, this complexity can be handled in many different ways. One example is to develop a central component that exactly knows how the aggregates are related and what to do if certain create/update/delete operations take place. Such a component could also be used to handle the integrity of the aggregates between many different databases for polyglot persistence. A further shortcoming with polyglot persistence is that more (disk) space is needed to store the aggregates in different databases. However, we think this is a trade-off between the low prices for disk space and computation time.

Furthermore, it is also possible with RDBMS to store aggregates in e.g. a BLOB. However, because of the ACID characteristics they exhibit a bad performance (Sadalage & Fowler 2012). Moreover, for reasons of brevity we exclude ACID characteristics and the CAP theorem.

Another research stream not covered by this paper is in-memory database (Loos et al. 2011; Plattner & Zeier 2012). These databases keep all their records in memory to perform faster queries. However, they are mostly based on the described databases. The SAP Hana in-memory database is, for instance, based on a column database (Plattner 2009). Additionally, quite all databases keep certain data structures in the memory to improve performance. Not surprisingly the idea to keep as much records as possible in memory is integrated in most of the currently implemented databases. In-memory databases however keep all records in memory.

7 CONCLUSION AND OUTLOOK

Business process management repositories manage large collections of process models containing thousands of models (Becker et al. 2012a; Dijkman et al. 2012; Yan et al. 2012). Additionally, they provide management functions like mining, querying, merging and variants management for process models. However, most current business process management repositories are built on top of relation database management systems (Yan et al. 2012), although this leads to performance issues. These issues result from the relational algebra, the impedance mismatch as well as new technological developments in the last 30 years as more and cheap disk and memory space, clusters and clouds. On the basis of our own experience as well as on the literature we have shown these issues in detail. To solve these problems we argued that we have to consider research about data modeling along database technologies as well as algorithm design and parallelization for the technology paradigms occurring nowadays. Furthermore, we have reviewed the related work and presented surveys revealing that the majority of BPMRs miss certain core repository functionalities or management techniques, required by practice (Shahzad et al. 2009; Yan et al. 2012). Subsequently, we summarized current paradigms based on NoSQL and algorithm design. Especially the work on aggregate oriented and graph databases is a very promising area for ideas that can be transferred to management techniques of BPMRs. Based on these paradigms we have shown how the performance of BPMR could be improved in terms of loading performance of processes (T_{Load}) and the computation of management techniques ($T_{Computation}$) resulting in an even faster application of such a technique: $T_{Management\ Technique} = T_{Load} + T_{Computation}$. These results have been reflected in an exemplarily application. The contribution for practice is avoiding pitfalls and issues while conceptually specifying or implementing BPMRs. Furthermore, the paper compiles references to the specification or detailed descriptions of the provided paradigms.

Future research will be conducted along two consecutive lines. First, we will evaluate the performance of document, key-value, column family and graph databases with different aggregates for different management techniques. Secondly, we will design efficient algorithms or reuse efficient algorithms for management functions and evaluate their parallelization potential. After we have chosen an appropriate database and have identified or developed algorithms for the management functions, we want to create an open-source implementation of a BPMR. This implementation should disseminate our findings and provide a platform for future (collaborative) research.

References

Awad, A. (2007). BPMN-Q: A Language to Query Business Processes. In Proc. of the 2nd Int. Workshop on Enterprise Modelling and Information Systems Architectures (EMISA'07) (Reichert, M. and Strecker, S. and Turowski, K. Eds.), 115–128, St. Goar, Germany.

Awad, A. and Sakr, S. (2010). Querying Graph-Based Repositories of Business Process Models. In Proc. of the 15th Int. Workshops on Database Systems for Advanced Applications (Yoshikawa, M.

and Meng, X. and Yumoto, T. and Ma, Q. and Sun, L. and Watanabe, C. Eds.), 33–44, Tsukuba, Japan.

Becker, J. and Breuker, D. and Delfmann, P. and Dietrich, H.-A. and Steinhorst, M. (2012a). A runtime analysis of graph-theoretical algorithms to detect patterns in process model collections. In Proc. of the 2nd Int. Workshop on Process Model Collections (La Rosa, M. and Soffer, P. Eds.), 31–42, Tallinn, Estonia.

Becker, J. and Delfmann, P. and Eggert, M. and Schwittay, S. (2012b). Generalizability and Applicability of Model-Based Business Process Compliance-Checking Approaches – A State-of-the-Art Analysis and Research Roadmap. Business Research, 5 (2), 221–247.

Beeri, C. and Eyal, A. and Kamenkovich, S. and Milo, T. (2006). Querying Business Processes. In Proc. of the 32nd Int. Conf. on Very Large Data Bases (VLDB '06) (Dayal, U. and Whang, K.-Y. and Lomet, D. and Alonso, G. and Lohman, G. and Kersten, M. and Cha, S.K. and Kim, Y.-K. Eds.), 343–354, Seoul, Korea.

Bernstein, P. and Dayal, U. (1994). An overview of repository technology. In Proc. of the 20th Int. Conf. on Very Large Data Bases (VLDB '94), 705–713.

Choi, I. and Kim, K. and Jang, M. (2007). An XML-Based Process Repository and Process Query Language for Integrated Process Management. Knowledge and Process Management, 14 (4), 303–316.

Dean, J. and Ghemawat, S. (2008). MapReduce : Simplified Data Processing on Large Clusters. Communications of the ACM, 51 (1), 1–13.

Delfmann, P. and Herwig, S. and Lis, L. and Stein, A. and Tent, K. and Becker, J. (2010). Pattern Specification and Matching in Conceptual Models - A Generic Approach Based on Set Operations. Enterprise Modelling and Information Systems Architectures, 5 (3), 24–43.

Dijkman, R. and La Rosa, M. and Reijers, H. A. (2012). Managing Large Collections of Business Process Models - Current Techniques and Challenges. Computers in Industry, 63 (2), 91–97.

Eid-Sabbagh, R. and Kunze, M. and Meyer, A. and Weske, M. (2012). A Platform for Research on Process Model Collections. In Business Process Model and Notation (Mendling, J. and Weidlich, M. Eds.), 8–22, Vienna, Austria.

Evans, E. (2004). Domain-Driven Design: Tackling Complexity in the Heart of Software. Addison-Wesley Professional, Amsterdam.

Hohpe, G. and Woolf, B. (2010). Enterprise Integration Patterns. Enterprise integration patterns: Designing, building, and deploying messaging solutions. Addison-Wesley, Boston.

Jin, T. and Wang, J. and Wu, N. and La Rosa, M. and Ter Hofstede, A. H. M. (2010). Efficient and Accurate Retrieval of Business Process Models through Indexing. In Proc. of the 2010 Int. Conf. on On the Move to Meaningful Internet Systems (OTM'10) (Meersman, R. and Dillon, T. and Herrero, P. Eds.), 402–409, Hersonissos, Crete, Greece.

Kleinberg, J. and Tardos, É. (2005). Algorithm Design. Pearson Education, Boston.

Leavitt, N. (2010). Will NoSQL databases live up to their promise? Computer, 43 (2), 12–14.

Libkin, L. (2003). Expressive power of SQL. Theoretical Computer Science, 296 (3), 379–404.

Lin, J. and Schatz, M. (2010). Design patterns for efficient graph algorithms in MapReduce. In Proc. of the 8th Workshop on Mining and Learning with Graphs (MLG 2010), 78–85, New York, USA.

Loos, P. and Lechtenbörger, J. and Vossen, G. and Zeier, A. and Krüger, J. and Müller, J. and Lehner, W. and Kossmann, D. and Fabian, B. and Günther, O. and Winter, R. (2011). In-memory Databases in Business Information Systems. Business & Information Systems Engineering, 3 (6), 389–395.

Plattner, H. (2009). A common database approach for OLTP and OLAP using an in-memory column database. In Proceedings of the 35th SIGMOD international conference on Management of data - SIGMOD '09 (Binnig, C. and Dageville, B. Eds.), 1–2, New York, USA.

Plattner, H. and Zeier, A. (2012). In-Memory Data Management. Springer, Berlin.

Sadalage, P. J. and Fowler, M. (2012). NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence. Addison-Wesley Longman, Amsterdam.

Sakr, S. and Awad, A. (2010). A framework for querying graph-based business process models. In Proceedings of the 19th international conference on World wide web - WWW '10, 1297–1300, Raleigh, North Carolina, USA.

Shahzad, K. and Andersson, B. and Bergholtz, M. and Edirisuriya, A. and Ilayperuma, T. and Jayaweera, P. and Johannesson, P. (2009). Elicitation of Requirements for a Business Process Model Repository. In Business Process Management Workshops (Ardagna, D. and Mecella, M. and Yang, J. Eds.), 44–55, Milano, Italy.

Shahzad, K. and Elias, M. and Johannesson, P. (2010). Requirements for a Business Process Model Repository: A Stakeholders' Perspective. In Business Information Systems (Abramowicz, W. and Tolksdorf, R. Eds.), 158–170, Berlin, Germany.

Skiena, S. S. (2008). The Algorithm Design Manual. Springer, London.

Stonebraker, M. and Madden, S. and Abadi, D. J. and Harizopoulos, S. and Hachem, N. and Helland, P. (2007). The End of an Architectural Era (It's Time for a Complete Rewrite). In Proc. of the 33rd Int. Conf. on Very Large Data Bases (VLDB '07), 1150–1160, Vienna, Austria.

Vom Brocke, J. and Becker, J. and Braccini, A. M. and Butleris, R. and Hofreiter, B. and Kapočius, K. and De Marco, M. and Schmidt, G. and Seidel, S. and Simons, A. and Skopal, T. and Stein, A. and Steiglitz, S. and Suomi, R. and Vossen, G. and Winter, R. and Wrycza, S. (2011). Current and Future Issues in BPM Research : A European Perspective from the ERCIS Meeting 2010. Communications of the Association for Information Systems, 28 (May), 393–414.

Yan, X. and Yu, P. S. and Han, J. (2005). Graph indexing based on discriminative frequent structure analysis. ACM Transactions on Database Systems, 30 (4), 960–993.

Yan, Z. and Dijkman, R. and Grefen, P. (2012). Business process model repositories – Framework and survey. Information and Software Technology, 54 (4), 380–395.